TEMA 4. Interpolación

Física Computacional I

Carolina Mendoza

Escuela Politécnica Superior
Universidad Nebrija

Unir n puntos usando un polinomio de interpolación con otro de grado $n-1$ que coincide con la cantidad de puntos que tenemos en el plano.

Dado cuatro puntos generar un polinomio de grado 3

x_i	0	0.2	0.3	0.4
p_i	1	1.6	1.7	2

$$p(x) = a_0x^3 + a_1x^2 + a_2x + a_3$$

$$a_0(0.0)^3 + a_1(0.0)^2 + a_2(0.0) + a_3 = 1.0$$
$$a_0(0.2)^3 + a_1(0.2)^2 + a_2(0.2) + a_3 = 1.6$$
$$a_0(0.3)^3 + a_1(0.3)^2 + a_2(0.3) + a_3 = 1.7$$
$$a_0(0.4)^3 + a_1(0.4)^2 + a_2(0.4) + a_3 = 2.0$$

Sistema a resolver

$$\begin{pmatrix} 0.0^3 & 0.0^2 & 0.0^1 & 1 \\ 0.2^3 & 0.2^2 & 0.2^1 & 1 \\ 0.3^3 & 0.3^2 & 0.3^1 & 1 \\ 0.4^3 & 0.4^2 & 0.4^1 & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} 1.0 \\ 1.6 \\ 1.7 \\ 2.0 \end{pmatrix}$$

El resultado es:

$$a_0 = 41.6667$$
$$a_1 = -27.5$$
$$a_2 = 6.8333$$
$$a_4 = 1$$

El polinomio de interpolación es:

$$p(x) = 41.6667 * x^3 - 27.5 * x^2 + 6.8333 * x + 1$$

La interpolación de Lagrange se utiliza si los datos no están equiespaciados.

$$p_n(x) = \sum_{i=0}^{n} L_i(x) f(x_i)$$

$$L_i(x) = \prod_{j=0,\, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}$$

Utilizando el mismo ejercicio anterior los dos primeros términos son:

$$L_0(x) = \frac{(x-0.2)(x-0.3)(x-0.4)}{(0-0.2)(0-0.3)(0-0.4)}$$

$$L_1(x) = \frac{(x-0)(x-0.3)(x-0.4)}{(0.2-0)(0.2-0.3)(0.2-0.4)}$$

Se construye el polinomio usando la fórmula para $f_n(x)$ para cada valor f_i,

$$p_3(x) = 1L_0(x) + 1.6L_1(x) + 1.7L_2(x) + 2L_3(x)$$

$$p_3(x) = 41.6667 * x^3 - 27.5 * x^2 + 6.8333 * x + 1$$

TEMA 4. Interpolación

sympy, matplotlib

```
import sympy as sym
import numpy as np
import matplotlib.pyplot as plt

x=sym.Symbol('x')
polisimple=3*x**2
px = sym.lambdify(x, polisimple)

pxi = np.linspace(1, 10, 100)
pfi = px(pxi)

plt.plot(pxi, pfi, 'o', label='Puntos')
plt.show()
```

De los vídeos y fotografías de la época se obtuvieron los siguientes datos:

t	0.00	0.15	0.30	0.45	0.60	0.75	0.90	1.05	1.20
x(t)	0.00	0.50	1.00	1.50	1.80	2.00	1.90	1.10	0.30
y(t)	0.00	4.44	8.88	13.31	17.75	22.19	26.63	31.06	35.50
z(t)	0.00	0.81	1.40	1.77	1.91	1.84	1.55	1.03	0.30

Agregando estas líneas al código de lagrange anterior, podemos tener una interpolacion en 3D.

- La interpolación de Hermite es una técnica para aproximar una función desconocida a partir de sus valores y las derivadas en ciertos puntos.
- Se puede realizar utilizando la base de Lagrange modificada para incluir información de derivadas.

Definición

La **interpolación de Hermite** es un tipo de interpolación polinómica que, a diferencia de la interpolación estándar, no solo se ajusta a un conjunto de valores de función en puntos dados, sino que también coincide con las derivadas de la función en esos puntos.

Como resultado se tiene un polinomio interpolante que es más suave y más representativo de la función original, especialmente cuando la función tiene cambios de dirección significativos.

Formalmente, dado un conjunto de $n+1$ puntos $\{(x_0, y_0, y_0'), (x_1, y_1, y_1'), \ldots, (x_n, y_n, y_n')\}$ donde y_i y y_i' son, respectivamente, el valor de la función y el valor de la primera derivada de la función en el punto x_i, el objetivo es encontrar un polinomio $H(x)$ de grado menor o igual a $2n+1$ que satisfaga:

$$H(x_i) = y_i \quad \text{y} \quad H'(x_i) = y_i' \quad \text{para todo} \quad i = 0, \ldots, n.$$

Dada una función $f(x)$ y sus derivadas $f'(x)$, $f''(x)$, ..., $f^{(k)}(x)$ evaluadas en los puntos $x_0, x_1, ..., x_n$, la interpolación de Hermite es dada por:

$$P(x) = \sum_{i=0}^{n} f(x_i) H_i(x)$$

Dados $(n+1)$ puntos distintos $\{x_0, x_1, \cdots, x_n\}$ denotado por m al valor: $m = 2 \cdot n + 1$ y siendo $f(x)$ una función de la que se conoce en cada punto x_i el valor f_i y su primera derivada f_i'
Se puedo encontrar un polinomio $p_m(x)$ de grado menor o igual que m verificando las $m+1$ igualdades siguientes:

$$\begin{aligned} p_m(x_i) &= f_i \\ p'_m(x_i) &= f_i' \end{aligned}$$

con $i = 0, \cdots, n$

Existe un único polinomio $p_m(x)$ que es solución.

$$p_m(x) = \sum_{i=0}^{n} f_i H_{i,0}(x) + \sum_{i=0}^{n} f_i' H_{i,1}(x)$$

donde:

$$\begin{aligned} H_{i,0} &= (1 - 2(x - x_i)L_i'(x_i))(L_i(x))^2 \\ H_{i,1} &= (x - x_i)(L_i(x))^2 \end{aligned}$$

siendo $L_i(x)$ el i-ésimo polinomo de base de Lagrange:

$$L_i(x) = \prod_{j=0,\, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}$$

TEMA 4. Interpolación

Código lectura de datos iniciales

```
sr_x = [(i * mat.pi) + (mat.pi / 2) for i in range(-5, 5)]
sr_fx = [mat.sin(i) for i in sr_x]
print(sr_fx)

deriv=np.zeros_like(sr_fx)
#Diferencias finitas
for i in range(1,len(sr_fx)-1):
    deriv[i] =(sr_fx[i+1]-sr_fx[i-1])/(sr_x[i+1]-sr_x[i-1])

# Hacia adelante primer punto
deriv[0] = (sr_fx[1] - sr_fx[0]) / (sr_x[1] - sr_x[0])
#Hacia atrás último punto
deriv[-1] = (sr_fx[-1] - sr_fx[-2]) / (sr_x[-1] - sr_x[-2])
```

Supongamos que queremos interpolar la función $f(x) = sin(x)$ en los puntos $x_0 = -5\pi$ y $x_1 = 5\pi$.

Utilizamos la fórmula de Hermite basada en Lagrange para encontrar el polinomio interpolante.

UNIVERSIDAD NEBRIJA

Aproximación Polinómica

Sea $a = x_1, x_2, \cdots, x_n = b$ se dice que el polinomio $s(x)$ es un spline de orden r si $s(x)$ es un polinomio de grado r en $[x_i, x_{i+1}]$, con $s(x) \in C^{r-1}$ en $[a, b]$

TEMA 4. Interpolación

librería

```
class scipy.interpolate.interp1d(x, y, kind='linear',
 axis=-1, copy=True, bounds_error=None, fill_value=nan,
 assume_sorted=False)
```

Los tipos pueden ser:

- 'linear',
- 'nearest',
- 'nearest-up',
- 'zero',
- 'slinear',
- 'quadratic',
- 'cubic',
- ...

TEMA 4. Interpolación

código

```
import numpy as np
from scipy.interpolate import interp1d

x = np.array([0, 1, 2, 3, 4])
y = np.array([0, 1, 4, 9, 16])

f = interp1d(x, y, kind='cubic')

# Evalúa el spline en el punto deseado x
x_new = 1.5
y_new = f(x_new)
print(y_new)
```

Griddata

`interp2d` es una función de la librería `scipy` que ha sido marcada como obsoleta. Ahora python utiliza `griddata` que está diseñada para la interpolación sobre una cuadrícula regular de 2 dimensiones. Permite estimar los valores de una función en puntos no dados inicialmente, basándose en los valores conocidos.